Comparing language models based on words, subwords and characters in different languages

Basak Eskili, Pascal Esser, Sindy Löwe (11731540, 11642858, 11594969)

Abstract

In this paper, we compare recurrent neural networks for language modeling based on word-, subword- and character-level inputs[1]. We employ a simple architecture consisting of 1,000 LSTM units on four different datasets, first the Penn Treebank and then the Harry Potter books in English, German and Turkish. Word-models have been shown to outperform character-based models in English. Nonetheless, we assumed that their advantage might not apply to languages with a richer morphology. However, our results disproved this assumption. Overall, the word-based model outperformed both the character- and subword-based models on all datasets, indicating that their relative performance does not depend on the characteristics of the language they are applied to but the characteristics of the model.

1 Introduction

In language modeling, it is more common to use words rather than characters as input since it has been shown to give more accurate results and low perplexity values in English (e.g. Inan et al. (2016)). However, for languages that have a rich morphology, this might not hold true. In German and Turkish, for example, many grammatical structures involve appending various endings to a word stem. This might make it difficult for the word-based model to learn the relationship between otherwise similar words and could lead to a worse performance. Character- or subword-based models are looking at fractions of words and might therefore be better at modeling word stems and grammatical endings (Mikolov et al., 2012). Additionally, they do not depend on a fixed vocabulary, which could enable them to predict infrequent words as well.

[1]Access our code on GitHub: https://github.com/pascalesser/nlp1_project

In order to test this hypothesis, we apply character-, subword- and word-based models on four datasets. First, we replicate the model described in Graves (2013) and apply it to the Penn Treebank dataset (Marcus et al., 1993). This provides us with a baseline and ensures that our implementation and comparison between the different models is correct. Since we want to test the performance of our models on different languages, we continue by training and testing them on a self-generated dataset consisting of all seven Harry Potter books (Rowling, 1997) written in English, German and Turkish. Finding a correlation between the morphology and the best atomic unit to use could provide us with a heuristic to improve language modeling in languages that are not studied as thoroughly as English.

The remainder of this paper is structured as follows: first we discuss related research (section 2) and give a formal definition of LSTM models (section 3). Then, we describe our approach, the dataset preprocessing, the model architecture and its training and testing procedure in section 4. Finally, we give an overview of our experimental results (section 5) and discuss them in section 6.

2 Related Work

There is extensive literature on the use of word-based models for predicting upcoming words. Starting from n-gram models (e.g. Brown et al. (1992)), where a fixed number of previous words is taken into account (typically between three and five) to more advanced approaches, such as neural network language models, like the ones presented in Bengio et al. (2003). Also, there is more recent work (e.g. Mikolov (2012)), that explores the use of more elaborate neural network architectures.

The second relevant line of work for our project comes from the use of character-based language models (e.g. Hochreiter and Schmidhuber (1997)). Here, we see a preference for using long short-term memory (LSTM) and gated recurrent units (GRU) models in recent work, as it allows each character to be predicted under the consideration of a longer history.

Bringing these two approaches together and comparing their strengths and weaknesses is the main point of this paper. Therefore we look at papers that work on the comparison of the two model types. The closest related to our approach are Graves (2013) and Kim et al. (2016). Graves (2013) uses simple LSTM models to compare a word- to a character-based approach and uses bits-per-character and perplexity as their evaluation metrics (see section 4.1.1). Overall, they are able to show that the word-based model has a better performance than the character-based model. Their analysis is limited to a comparison on an English datasets, namely the Penn Treebank.

Kim et al. (2016) takes a similar approach but uses a more advanced network architecture using convolutional layers in addition to LSTMs to construct word- and character-based models. Interestingly, it finds contrary results to Graves (2013), as they are able to show that the word-based model performs worse than the character-based model. They apply their models to a number of different languages consistently achieving a better performance when using the character-based model.

In order to test, if the performance of word- and character-based models is indeed language depended we rebuild the model by Graves (2013) and apply it to different languages.

3 LSTM Networks

In this project, we use recurrent neural networks (RNN). This section is aimed at providing the reader with a short introduction to this special class of neural networks. The most important difference to the traditional feed-forward neural network is that it includes directed cycles between nodes. Therefore, in addition to the current input, its output from the previous time-step is fed back into it as well. This makes it possible to take previous inputs into account. A very simple representation can be found in Figure 1, which displays the recurrent nature of feeding the output back into the node unfolded over time.

Starting from this simple version of a RNN, where the hidden layer is an element-wise func-

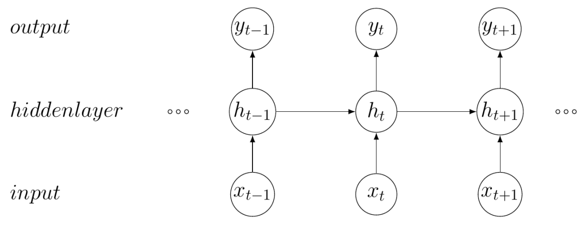

Figure 1: For a better understanding of a simple RNN model, consisting of the output, hidden and input layer. The depicted network is unfolded over three time-steps ($t-1$, t, $t+1$). Additionally to the current input, the hidden layer also receives its own output from the previous time-step to calculate the output for the current time-step.

tion over a non-linearity, we can add LSTM blocks (Figure 2) to avoid the vanishing gradient problem. This gives us a LSTM network (Gers et al., 1999), the network architecture used in this paper.

For the LSTM block, we first define the following variables for the time step t for a hidden layer dimension h and input dimension d: $\boldsymbol{x}_t \in \mathbb{R}^d$ is the input vector for the LSTM block. $\boldsymbol{c}_t \in \mathbb{R}^h$ is the cell state vector, storing information over time in the cell. $\boldsymbol{f}_t \in \mathbb{R}^h$ is the forget gate activation vector. It takes a look at the output of the hidden unit of the previous time step and returns a number between 0 and 1. Here, zero means that it forgets all the previous information and one to keep all of it. $\boldsymbol{i}_t$ is the input gate activation vector and is used to update the state given the current context. $\boldsymbol{o}_t \in \mathbb{R}^h$ is the output gate activation vector, which provides information about the current state that might be important to have for the next input. $\boldsymbol{h}_t \in \mathbb{R}^h$ is the output vector of the LSTM block. Finally $\boldsymbol{W} \in \mathbb{R}^{h \times d}$ and $\boldsymbol{b} \in \mathbb{R}^h$ are the weight matrix and the bias vector, where $\boldsymbol{W}_{xy}$ would give the weights from x to y. Formally, the calculations within a LSTM cell are defined as follows:

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{cf}\boldsymbol{c}_{t-1} + \boldsymbol{b}_f)$$
$$\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ci}\boldsymbol{c}_{t-1} + \boldsymbol{b}_i)$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{co}\boldsymbol{c}_{t-1} + \boldsymbol{b}_o)$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tau(\boldsymbol{c}_t)$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \tau(\boldsymbol{W}_{xc}\boldsymbol{x}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c)$$

where $\circ$ is the Hadamand product, $\sigma(\cdot)$ is the logistic sigmoid function and $\tau(\cdot)$ the hyperbolic tangent function. The weights and biases are learned during training and for the initialization we set $\boldsymbol{c}_0 = \boldsymbol{0}$ and $\boldsymbol{h}_0 = \boldsymbol{0}$.

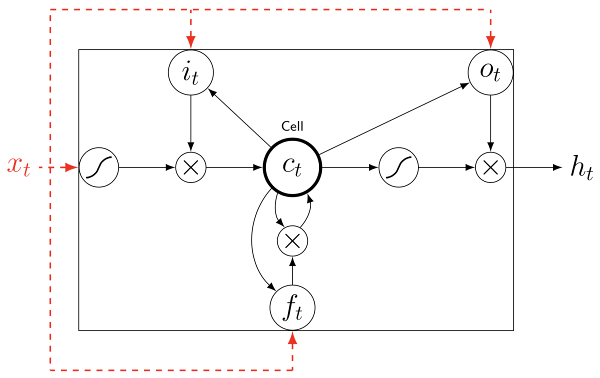

Figure 2: Layout of a single LSTM cell. x_t is fed into the block at four points: directly over a nonlinear function (sigmoid), into the forget gate vector f_t, the input gate vector i_t and in the output gate vector o_t. Furthermore c_t represents the cell state vector. The formal representation can be found above in section 3.

4 Approach

In this section, we describe the methods we used in order to test how word-, subword- and character-based models compare in different languages. First, we depict the datasets we used and how we preprocessed them. Then we characterize the architecture of our neural network and its learning procedure. In the end, we explain the particularities of our testing procedure and how we can use our models in order to generate text.

4.1 Datasets

In our experiment, we start training our different models on the Penn Treebank dataset in order to compare our implementation to Graves (2013). Then, we want to test the performances of our models in different languages. To that end, we use a self-generated Harry Potter dataset which is a collection of all seven books in English, German and Turkish. We chose Harry Potter as a dataset, because it is available in the three languages that we would like to study and because a comparison between languages might be facilitated by the fact that we work on translations based on the same storyline.

4.1.1 Preprocessing

During preparation of the dataset, we take the Penn Treebank dataset as a reference, since its format is widely accepted. The dataset is separated in such a way that there is only one sentence per line. Every word is lowercased so that case sensitivity will not effect the model while training. Addition-

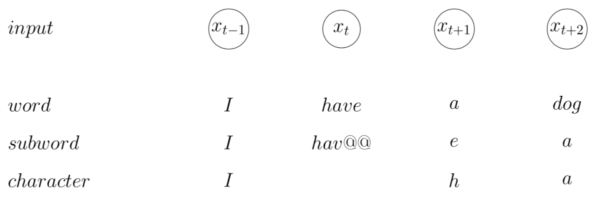

Figure 3: Here, we see an example input for the character-, subword- and word-based models (see section 4.1.1) for the string "I have a dog" over the timesteps $t-1$, t, $t+1$ and $t+2$. For the word-based model all four words are feed into the model over four time steps. For the subword-based model "have" is split into "hav@@ e". For the character-based model, it is important to note that at time t a blanc space is fed into the network, whereas blanc spaces are included implicitly in the word- and subword-based models.

ally, all symbols except for **,** (comma), **'** (apostrophe) and **.** (dot) are removed and numbers are replaced by **N**.

We create the dataset with the unknown tag by replacing infrequent words with the $< unk >$ token based on a list of the 10,000 most common words.

In addition to this dataset, we also create a subword version of each dataset by segmenting tokens into subword units, in which each unit contains three or less letters. Subwords that are within a word are concatenated with two @ symbols in order to distinguish them from word-ending subwords. For this process, we make use of a public repository that is provided for neural machine translation purposes (Rico Sennrich, 2017).

In order to test our hypothesis, we use three different atomic units to feed to our models. Figure 3 shows how the word, subwords and characters are used as inputs in our models respectively.

For each language, the dataset is divided into three parts for training, validation and testing. We do so by shuffling the lines (sentences) in the text files and selecting them randomly. 80 % of the data is used for training, 20 % of the data is divided equally for testing and validation.

Table 1 gives the most important statistics about our employed datasets. We can see that the overall word count is comparable over all languages, but Turkish has a richer vocabulary compared to German and English. Thus, it is effected the most by replacing out-of-vocabulary (OOV) words with the unknown tag. Additionally, the word length is the longest for Turkish, which might influence its perplexity scores. The Penn Treebank has the smallest amount of sentences, while having about the same number of words as the Turkish Harry Potter dataset, which implies that it consists of longer sentences.

Table 1: Statistics for the four datasets Penn Treebank (Penn), English (EN), German (GE) and Turkish (TR) Harry Potter. $|w|$ indicates the total number of words, |VOC| the size of vocabulary, $\overline{w}$ the average word length and $|sent|$ the total number of sentences.

	$\|w\|$	|VOC|	$\overline{w}$	$\|sent\|$
Penn	984,738	35,617	4.62	43981
EN	1,333,649	22,690	4.05	63841
GE	1,351,296	44,363	4.55	63049
TR	919,642	78.744	6.00	66885

In Table 2, the number of different atomic units within our four datasets is depicted. While the number of different words is held constant at 10,000 by replacing infrequent words with the $< unk >$ token, the number of subwords varies between the datasets due to the generation procedure we employ. We use the dataset that already includes the $< unk >$ token in order to create the subword dataset. Since the vocabulary in Turkish is much bigger, it will include more of these tokens and as a consequence a lower number of subwords. Additionally, the number of different characters is held constant at 35 for our self-generated datasets, but equals 50 in the Penn Treebank. This is due to the fact that it includes rarely occurring symbols such as $, \ and /.

Table 2: The number of different words, sub-words and characters in the four datasets Penn Treebank (Penn), English (EN), German (GE) and Turkish (TR) Harry Potter.

	word	subword	character
Penn	10,000	3,987	50
EN	10,000	3,555	35
GE	10,000	3,271	35
TR	10,000	2,649	35

4.2 Neural Network Architecture

We employed the same neural network architecture as described in Graves (2013). The input is given to the network in the form of one-hot-vectors, whose length is equal to the number of different atomic units to be trained. First, we apply a single hidden layer consisting of 1,000 LSTM units. Then we use a fully-connected layer to decode the outputs of the LSTMs such that they match the number of atomic units in our dataset.

The size of this fully-connected layer is influenced by the number of different atomic units that we train on. Therefore, this architecture yields a total number of approximately 54 million parameters for the word-based models and approximately 4.2 million parameters for the character-based models. For the subword-based models the numbers vary due to different amounts of subwords present in the datasets. For the Penn Treebank dataset, we obtain the biggest model containing approximately 23.9 million parameters, for the English Harry Potter dataset there are 21.8 million parameters, 20.4 million for the German Harry Potter and 17.2 million for the Turkish dataset.

Since there is quite some discrepancy between these numbers, we try to account for this by testing a second version of the character-based model consisting of 2,000 instead of 1,000 LSTM units. This increases the number of parameters to approximately 16.4 million and should therefore make the comparison between the model types fairer.

We use the softmax function to transform the output of our network into probabilities for the different words, subwords or characters:

$$f_i(x) = \frac{exp(x_i)}{\sum_j exp(x_j)}$$

In order to evaluate the performance of our network during training, we use the negative log likelihood loss:

$$loss(x_i) = -log(f_i(x))$$

4.3 Training Procedure

We use stochastic gradient descent for the training of our model. We start with a learning rate of 20, which is divided by four whenever the validation loss did not improve after one epoch of training. We do not use weight decay or momentum, but apply clipping to the norm of all gradients such that it is always equal to or smaller than 0.25.

Each batch consists of 20 sequences which contain 35 words, subwords or characters each. After each batch, the hidden states of the network are wrapped into new variables which detaches them from their history. At the end of each epoch, we additionally reset the hidden state of the neural network.

Overall, we train the word- and subword-based models for 15 epochs and the character-based model for 20 epochs, as its validation loss was not stagnating yet.

4.4 Testing Procedure

Next, we will describe the details of our testing procedure. First, we describe the technique of dynamic evaluation, before depicting the evaluation metrics we employed.

4.4.1 Dynamic Evaluation

Usually, the weights are fixed when applying a neural network on the test data. In our case, however, the inputs are the targets that we were trying to predict in the previous time-step. It is therefore legitimate to allow the network to adapt its weight during evaluation, if it only gets to see the test data once. This method was introduced as dynamic evaluation in Mikolov (2012) and is described in more detail in Krause et al. (2017). It was also used to generate the results in Graves (2013) that we aim to replicate.

In order to evaluate the effect of this technique, we additionally test our models without it by keeping the weights fixed when testing.

4.4.2 Performance measurements

In order to evaluate the performance of our models, we use two measurements. First, we employ perplexity (PPL), which is commonly used for the assessment of language models and tells us how well our models are at predicting the next word in a text. We calculate it by using the output of our loss function, the negative log likelihood L_e. As the logarithm here is calculated to the base e, we can compute the perplexity as follows:

$$\text{PPL} = e^{L_e \cdot \overline{n}}$$

Where $\overline{n}$ is the average number of atomic units that the words in the test set are split into. For the word-based model this is one, for the subword-based models the values range between 1.69 and 1.95. For the character-based models we need to add one to the average word length in order to account for the fact that it also needs to predict spaces explicitly. This yields values between 5.05 and 7.02 .

As a second measurement we use is bits-per-character (BPC), which describes the average character-wise cross-entropy. For the character-based model this is equal to the negative log likelihood L_2 to the base 2. For the subword- and word-based models, we need to break this value down to apply to single characters. This is achieved by:

$$\text{BPC} = \frac{L_2 \cdot \overline{n}}{\overline{w} + 1}$$

Where $\overline{w}$ is the average word-length in the test-set. We add one to this value, since the spaces after words need to be predicted in a character-based approach as well.

4.5 Generative Model

In order to generate new text using our trained language models, we feed a single random word, subword or character which was present in the training dataset into the network. Based on this, the network will output a prediction for the most likely word, subword or character to follow. We feed this output back into the network and keep repeating this process until a certain amount of text has been generated. The results of this process can be seen in A.2 and A.3, where we generated sentences using the word- and character-based models which were trained on the English Harry Potter dataset.

5 Results

In this section, we will describe the results of our experiments. First, we will give the achieved perplexities of the character-, subword- and word-based models on the different datasets. Then we investigate whether increasing the size of the character-based model can boost its performance.

5.1 Main findings

In Figure 4 we can see the performances of our models on the four different datasets measured in perplexity. Additionally, we have listed these values and the bits-per-character scores in Table 3. First of all, we can see that we were able to replicate the results on the Penn Treebank given in Graves (2013). Interestingly, we can see that even though our BPC values for the character-based approach are the same, our results indicate a slightly higher perplexity. Furthermore, we were able to achieve better results on the word-based approach.

When comparing the results for the basic approach to the dynamic evaluation, we can see that adjustable weights during test time can have a tremendous effect on the overall performances. For the subword-based model applied on the

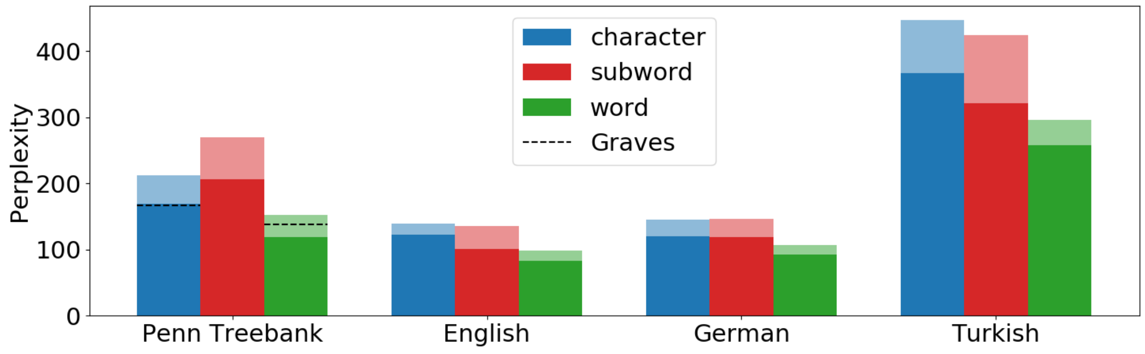

Figure 4: Perplexities for the character-, subword- and word-based models (in blue, red and green) for the four tested datasets Penn Treebank and the English, German and Turkish Harry Potter. Solid bars represent the performance when dynamic evaluation is applied, the shaded areas depict the perplexities obtained when keeping the weights fixed during testing. The results of Graves (2013) are shown as dashed lines.

Turkish Harry Potter dataset, dynamic evaluation yields the biggest enhancement with a relative performance gain of 34%.

Table 3: Bits-per-character (BPC) and perplexity (PPL) measurements for the character-, subword- and word-based models on the four different datasets. We tested two approaches, B stands for the basic approach using the trained model on the test data without adjusting the weights, the DE approach uses dynamic evaluation. The results of Graves (2013) are given for comparison.

dataset	appr.	character		subword		word	
		BPC	PPL	BPC	PPL	BPC	PPL
Graves (2013)	DE	1.32	167	—	—	1.27	138
Penn	B	1.37	212	1.44	269	1.29	153
	DE	1.32	169	1.37	206	1.23	119
English	B	1.41	139	1.40	136	1.31	98
	DE	1.37	122	1.32	101	1.26	83
German	B	1.29	145	1.30	147	1.21	106
	DE	1.25	120	1.24	119	1.18	92
Turkish	B	1.25	447	1.24	424	1.17	296
	DE	1.21	367	1.19	321	1.14	258

Overall, the word-based model achieves the best results on all datasets both with the basic and dynamic evaluation approach. For the English and Turkish datasets, the subword-based model shows slightly better performances than the character-based model. In the German dataset, the two models achieve comparable results. The biggest difference between these models becomes evident for the Penn Treebank dataset. Here, the character-based model achieves a considerably better performance.

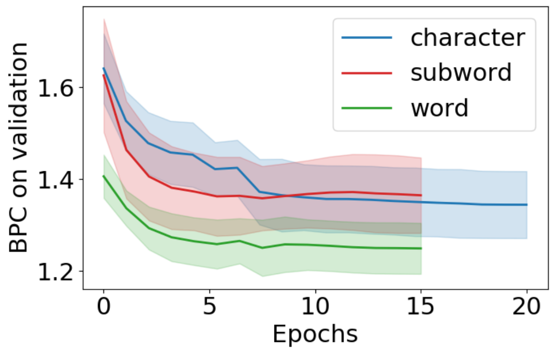

Figure 5: Bits-per-character (BPC) averages and standard deviations during training for the character-, subword- and word-based models on the four datasets.

In order to further investigate the behavior of our models, we also plotted the average BPC scores that the models obtained on the validation sets of all datasets during training (see Figure 5). We plot the BPC, because it describes the achieved loss but is normalized over the length of the atomic units the different models are trained on and is therefore comparable between the approaches. In the plot, we can see that averaged over the datasets, the word-based model achieved the lowest and therefore best BPC score, while the character-based model has a slight advantage over the subword-based model. It is also evident that the subword-based models overfit on the data, as we can see an increase in the BPC values around epoch ten. Such a problem does not seem to arise for the other two models however, as their performances on the validation sets increase throughout training.

5.2 Large character-based model

So far, we have seen that the word-based model outperforms the character-based model in all respects. The comparison we have done might be unfair, however, as the word-based model has approximately 13 times more parameters. In order to account for this difference, we train a second, larger character-based model. Here, we increase the number of LSTM units in the network from 1,000 to 2,000, expanding the model to have approximately four times the number of parameters. The results for this model can be seen in Table 4.

Table 4: Bits-per-character (BPC) and perplexity (PPL) measurements for the two versions of the character-based model (character L stands for the large character-based model using 2000 LSTMs). Again, we tested two approaches, B stands for the basic approach using the trained model on the test data without adjusting the weights, the DE approach uses dynamic evaluation.

dataset	appr.	character		L character	
		BPC	PPL	BPC	PPL
Penn	B	1.37	212	1.39	224
	DE	1.32	169	1.31	166
English	B	1.41	139	1.42	145
	DE	1.37	122	1.37	119
German	B	1.29	145	1.31	155
	DE	1.25	120	1.25	121
Turkish	B	1.25	447	1.28	505
	DE	1.21	367	1.22	379

We can see that increasing the model size actually leads to a worse performance for all datasets when using the basic approach with fixed weights during testing. Interestingly, applying dynamic evaluation leads to a bigger performance gain for the larger model, which can therefore catch up and produce similar results to the smaller model.

6 Discussion and Conclusion

As already discussed, Kim et al. (2016) was able to show that with their approach a character-based model performs better than the word-based models across different languages. Nevertheless, they acknowledge that there is a difference in modeling distinct languages. They mostly differentiate between English and more morphologically rich languages like German or Russian. This lead to our hypothesis that these differences in languages might be enough for the model designed by Graves (2013), which shows that word-based models outperform character-based models in English, to perform better with a character-based approach when applied on morphologically rich languages like German or Turkish.

The first step of our project was to replicate the results described by Graves (2013). We were able to show that our models perform similarly on the Penn Treebank dataset and we even managed to beat the perplexity scores of Graves (2013) for the word-based model when using dynamic evaluation.

The differences in performance between Graves (2013) and our implementation can be explained by the fact that our training parameters might be different as they were not fully disclosed in Graves (2013). Additionally, we used different rounding to calculate our results. For example, Graves (2013) cut the average word length for the Penn Treebank dataset after the first decimal, while we used the whole float number, leading to slightly worse perplexity scores in our experiments.

When we take into account the statistics of our datasets, we can see that datasets with properties indicating a more difficult use of language lead to worse performances of our models. The Penn Treebank dataset was generated based on the Wall Street Journal, a English-language international newspaper focused on business, whereas Harry Potter is a fantasy novel, aimed at a large group of (mostly) young readers. Therefore, we expect the language used in Harry Potter to be easier and in turn this leads to better performances of our models on this dataset for English and German. Turkish, however, shows much more complicated characteristics, it has a much bigger vocabulary size and a longer average word length. These properties are reflected in our results as well, as the bits-per-character and perplexity scores we achieve on this language are inferior.

In general, we can see that the word-based model performs the best across datasets, followed by the subword-based model. Overall, the character-based model shows the worst results, but overtakes the subword-model on the Penn Treebank dataset. That might be due to the fact that this dataset includes the highest number of different subwords, whereas there is a constant number of different atomic units for the word- and character-based models. Following this explanation, the subword-model should perform the best on the Turkish dataset as it includes the lowest number of different subwords. This, however, does not hold. We postulate that the inferior performance on the Turkish dataset can be explained by the high out of vocabulary (OOV) rate in this dataset. Even though the number of different subwords might be lower, the text might include so many $< unk >$ tokens, that it becomes hard to model the overall context within sentences.

The word-based model outperformed the character-based model. When looking at these results, however, we have to take into account that the character-based model is much smaller in terms of parameters being approximately 13 times smaller than the word-based model. Therefore, in order to make for a fairer comparison, we increase the number of parameters for the character-based model by using 2,000 instead of 1,000 LSTM units. What we observe is that it did not enhance the performance, but actually makes it worse when using the basic approach with fixed weights during testing. When applying dynamic evaluation both versions of the character-based model achieve similar results, indicating that the larger model simply overfits during training. This result indicates that the word-based model might outperform the other two models due to the larger fully-connected layer that maps the output of the LSTMs to the vocabulary and not the raw count of parameters in the model itself. Further research is necessary to determine whether increasing the size of the fully-connected layers in the character- and subword-based models can enhance their performance.

As Graves (2013) only offers results for employing dynamic evaluation, we also extended our testing procedure and used this technique for further optimization. Since training on the test set and therefore not keeping a strict separation between training, validation and test sets is very unusual, we decided to clearly distinguish our results achieved with dynamic evaluation to those obtained with fixed weights. We can observe that dynamic evaluation improves the results for character-, subword and word-based models over all tested languages.

All in all, we investigated the performance of character- and word-based LSTM language models on three languages, English, German and Turkish. In order to enable us to make a better comparison, we additionally test a subword-based model for each language. We showed that our initial hypothesis does not hold, as both the character-based and the subword-based models perform worse than the word-based model on all datasets tested. Taking into account the results of Kim et al. (2016), who showed that the character-based model outperforms the word-based model across various languages, we conclude that the relative performance of the different approaches is indeed not language-, but architecture-dependent. Further research is necessary to confirm this finding.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3(Feb):1137–1155.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics* 18(4):467–479.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with lstm .

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* .

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word classifiers: A loss framework for language modeling. *CoRR* abs/1611.01462. `http://arxiv.org/abs/1611.01462`.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *AAAI*. pages 2741–2749.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. 2017. Dynamic evaluation of neural sequence models. *CoRR* abs/1709.07432. `http://arxiv.org/abs/1709.07432`.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.* 19(2):313–330. http://dl.acm.org/citation.cfm?id=972470.972475.

Tomáš Mikolov. 2012. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April* .

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. 2012. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)* .

Diogo Mesquita Ahmed Elgohary Rico Sennrich. 2017. Subword neural machine translation https://github.com/rsennrich/subword-nmt.

J. K. Rowling. 1997. *Harry Potter and the Philosopher's Stone*, volume 1. Bloomsbury Publishing, London, 1 edition.

A Supplemental Material

A.1 Team responsibilities

Most of the work was done in team meetings and therefore ensuring that the workload was evenly distributed between the authors. In general, Basak Eskili focused on the preprocessing and generation of the data, Sindy Löwe on the implementation of the presented models (which included some additional workload) and Pascal Esser on the theoretical part for building the models.

A.2 Generated Text from the word-based model trained on the English Harry Potter dataset

```
he turned back to himself .

right , he said , looking up at
    fudge 's face .

i m staying here harry muttered
    to ron .

when she shook her head again .

sleep , dudley was in the library
    , harry 's thoughts would not
    be revealed in grimmauld
    place .

harry got himself , but before
    somebody open external 's eye
    , does that malfoy or thinner
    like him as it did , as the
    elf 's stone set ever mended
    with <unk> toffee cloaks .

have a good time , harry .

only they could move through the
    canopy doors .

what 's that seeing a job like he
    did subsided i was wearing a
    dog deemed .

snape came striding into the room
    into night .
```

A.3 Generated Text from the char-based model trained on the English Harry Potter dataset

```
authority that scrimgeour put
    them to the tent in a chain of
    judgment.

he was deserted wearily to its
    mentioned besk.

i colls with people to stay like
    potter , you re supposed to be
    asked up onto the shop.

these words has run of escape
    from fudge.

while hagrid let go of him , and
    harry heard voldemort 's
    fourgestush next to the
    village , which linus were
    looking
unbreasable.

crabbe has n't had to make sure ,
    ron ignored her.

the thick during a very old and a
    <unk> of crunching ponstic
    them <unk> with all his empty
    passage.

transforming dream about
    dumbledore too , or weasley ,
    said fred , peering high above
```

the teenage stillness .

glancing toward their teapons the man 's fall <unk> , but none strode off he saw neville rather thin a single woman shouted , there were no rustle hammering .

how do they fail torture for us or something .

good day i m going to happen and ginny harry harry yelled behind them .

his voice was still full if he had met his hands this manquare on the elf 's laughter on it .